Deerly Departed

Problem Solving Technique: Solve an easier version first
It can be easier to reckon with, and might give you insight to the full version of the problem.

Always do the subtasks first. Even then, don't feel limited to needing subtasks—make up your own!

Consider the case where $x_i = y_j = 1$ always, which implies that $n = m$.

In this case, the task is to simply figure out how to *match up* the deer and caretakers, such that the happiness is maximized (or minimized).

Problem Solving Technique: Play
Play around with actual concrete cases on pen and paper. See if there are any patterns you can notice among them.

After playing around with concrete values, you should get start to get this feeling...

Of course we want to pair big numbers with other big numbers if we want the sum maximized. And *of course* we try to pair the big numbers with the small numbers from the other list, if we want the sum minimized.

Maybe we should take that to its logical extreme.

Standard Approach: Extreme principle
Trying to grapple with a problem? Look at "extreme" cases and see how they behave.

Claim
In the case where $x_i = y_j = 1$ always, the happiness is *maximized* when c and d are both sorted; and *minimized* when one is sorted, and the other is sorted in reverse order.

Remark
This is known as the Rearrangement Inequality, and is somewhat standard in both contest math, and competitive programming (as an example of a "greedy" approach).

The proof of the Rearrangement Inequality is deferred until later.

How to solve the original version of the problem?

Useful Trick: Solve a looser version of the problem.
Consider a looser version of the original problem (i.e. it admits *more* possible solutions). If the answer in the looser version is also always valid in the original, then that solves the original problem as well!

Claim
Even in the full version of the problem, the happiness is *maximized* when c and d are both sorted; and *minimized* when one is sorted, and the other is sorted in reverse order.

Proof
Suppose the deer and caretakers *don't* need their time scheduled contiguously. That is, suppose we are allowed to break up everyone's times into 1-unit fragments, and schedule those fragments however we want throughout the day (the fragments don't *have* to be chunked together).

In this looser version, the answer is given by the Rearrangement Inequality: Sort the c fragments, and either sort d the fragments (to maximize happiness) or reverse-sort them (to minimize happiness).

But in a sorted order, the 1-unit fragments of some deer (or caretaker) will end up next to each other anyway. So this must also be the optimal solution in the case where each deer and caretaker must have their time scheduled contiguously!

Implementation

To actually *compute* the sums in $O(n \lg n)$ time, you would have to implement some kind of line sweep algorithm (generally considered standard in comp prog). You can ask the Discord server for details!

Implementation

For C++ users, *note that the answer can be large.* We can show that the maximum value of $\approx (10^6 \cdot 10^6) \cdot (10^5 \cdot 10^6)$ does not fit in a 64 bit data type, but it *does* fit in a 128 bit integer like `__int128`.

The data type `__int128` is not supported by `cout`, but it is not too hard to manually output it digit-by-digit.

Proof: Rearrangement Inequality

The proof is by exchange argument. WLOG suppose c is sorted (so $c_i \leq c_j$ for all $i \leq j$), and let D_i be the value of d paired with c_i.

Suppose there exists $i < j$ such that $D_i > D_j$. Then, we can always make the sum bigger (or at least not worse) by swapping i and j, as

$$c_i D_i + c_j D_j \leq c_j D_i + c_i D_j.$$

We can prove this by starting at,

$$c_i \leq c_j,$$

and so we can multiply the same (positive) value to both sides:

$$c_i(D_i - D_j) \leq c_j(D_i - D_j)$$

which resolves to the desired inequality, after doing some algebra.

A pair (i, j) such that $i < j$ but $D_i > D_j$ is called an *inversion*. There is only one way to get a sequence with no inversions—if D is sorted. Since removing an inversion never decreases the happiness, it follows that the sorted D has a happiness $\geq$ the happiness of all other orderings.

Minimization works by a similar argument.